SN Computer Science (2022) 3: 405
https://doi.org/10.1007/s42979-022-01288-6

ORIGINAL RESEARCH

Real-Time Heuristic-Based Detection of Attacks Performed on a Linux Machine Using Osquery

Sarfaraz Ahamed[1] · Ramanathan Lakshmanan[2]

Received: 29 April 2022 / Accepted: 1 July 2022 / Published online: 28 July 2022

Abstract
With the increase in Unix-based operating system for web servers and IoT devices, it has become crucial to detect attacks that are performed on these critical devices. Detection can be done at multiple layers of David Bianco's Pyramid of Pain [http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html] which consists of the following layers: TTPs, Tools, Network/Host Artifacts, Domain Names, IP Address, and Hash Values. As majority of recent work focuses on machine learning to help detect attack, our focus of this paper is detection of attacks predominantly at the TTPs, Tools, and Network/Host Artifacts levels using heuristic-based detection. This will allow us to provide detection in depth to machine learning models by detecting known bad that is sometimes missed by machine learning models. Using osquery, we were able to create a real-time heuristic-based detection script for Linux. This script takes in each log from the osquery and tries to match against various conditions to detect initial connections, lateral movement, and privilege escalation.

Keywords Heuristic · Real-time detection · Unix · Attacker's behaviour · Host-based intrusion detection · HIDS

Introduction

The internet has been built on trust; if each device and user do what they are allowed to do, then there would be no breach of trust and thus no need of the field of cybersecurity. It is when this trust is broken and misused, there is a need for detection, prevention, policies, and more. This trust is broken by attackers that are driven by various kinds of motivation. Some of them are for money, to show others that can control other's machine, and for political and ideological reasons. There has been a rise in the use of Unix-based system, but, respectively, there has not been a growth in detection and response solutions for Unix-based system, this has caused those who break the trust to go about undetected. Before moving into how these attackers who break the trust can be detected, let's see the two types of attacks that these attackers perform on a Unix machine.

The first kind of attack is network-based attacks. In this type of attack, attackers try to break the trust using different types of network protocols. One of the most common types of attack in this category is brute-forcing. The attacker might try to exploit known vulnerabilities to get access to the machine, brute-force web login pages, open RDP ports, open ftp ports, open SSH ports, and more. Using brute-forcing the attackers want to get inside the network and machine where they have no access being in, thus breaking the trust. Another kind of attack that is performed is Denial of Service (DoS) and Distributed Denial of Service (DDoS). This kind of attack is done by opening multiple connections to any service, in a limited period, than what the service can handle. This cause the service to be not accessible for other users, thus the name.

The second kind of attack is endpoint-based attack. In this type of attack, the attackers that get access into some unauthorized machine can perform some tasks as the users of the machine. This requires that they have access to the machine in the first place. This kind of attack includes accessing

This article is part of the topical collection "Predictive Artificial Intelligence for Cyber Security and Privacy" guest edited by Hardik A. Gohel, S. Margret Anouncia and Anthoniraj Amalanathan.

✉ Ramanathan Lakshmanan
lramanathan@vit.ac.in

Sarfaraz Ahamed
sarfarazahamed2k@gmail.com

[1] Department of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India

[2] Department of IoT, Vellore Institute of Technology, Vellore, India

private information stored in the system, escalating their privileges to get access into a user with more permission, exfiltrating sensitive information, and more.

Modern research on host-based intrusion detection systems focusses primarily on machine leaning techniques to identify and detect malicious activity. In papers [1–5] discussed below, machine learning models are used to create a model using datasets created to provide a way to detect malicious activity. This approach is very useful in detecting threats, but it does not give a detection rate of 100%. Hence an approach of detection in depth can be used with machine learning-based detection and heuristic-based detection. This approach of detection in depth allows each model to detect what each model is known best to detect; heuristic-based detection for detecting known bad and machine learning models for detection new variations of attacks.

Syslog does not provide sufficient visibility into a Unix machine to detect initial connections, privilege escalation, and lateral movement. To get sufficient visibility, osquery from Meta is a good solution. Osquery is an operating system instrumentation agent that provides a unique and refreshing approach to security. It delivers a single-agent solution using a universal query language to collect rich datasets for multiple use cases. Osquery simplifies the process of understanding your infrastructure by exposing an operating system as a high-performance relational database [6]. Using osquery we can schedule queries that will run at a fixed interval of time to query the operating system for specific actions performed in it, so that we can detect attacker's behaviour, both network-based attacks to and from the victim machine and also endpoint-based attacks in the machine itself.

Related Work

Literature Review

The literature review of datasets is from the previous 10 years. There were 4 datasets that were created from 2012 for detecting attacks. From the 4 datasets, 3 of the datasets focus on detecting network-based attack and one dataset focuses on detecting endpoint-attacks based on system trace call.

Dataset Name: CSE-CIC- IDS-2018 [7, 8]

Created by: University of New Brunswick

Year Created: 2018

Description: This dataset consists of 80 features with a wide range of attacks such as Brute force, Portscan, Botnet, Dos, DDoS, Web attacks, and Infiltration. The network traffic was captured for the duration of capture of 5 days with an attack infrastructure of 4 PCs, 1 router, 1 switch and a victim infrastructure of 3 server, 1 firewall, 2 switches, 10 PCs. Network profiles were used to generate the dataset in a specific manner, hence as networks architectures are different in how they are setup and used in different organizations, the behavioural patterns of network assaults are shown here in the year of 2018. As network assaults change over time, this dataset was created to cover the techniques that were performed in the year of 2018 and as an inundation from the previous year (2017). A limitation of this dataset is that the data samples created by network flow analysis are saved in files, and processing these files is a time-consuming activity since each file contains a significant number of data instances.

Dataset Name: CSE-CIC-IDS-2017 [8, 9]

Created by: University of New Brunswick

Year Created: 2017

Description: This dataset consists of 80 features with a wide range of attacks such as Brute force, Portscan, Botnet, Dos, DDoS, Web attacks, and Infiltration. The network traffic was captured for the duration of capture of 10 days with an attack infrastructure of 50 PCs and a victim infrastructure of 420 PCs, 30 servers. Network profiles were used to generate the dataset in a specific manner, hence as networks architectures are different in how they are setup and used in different organizations, the behavioural patterns of network assaults are shown here in the year of 2017. As network assaults change over time, the following year another dataset was created to cover the techniques that were performed in the year of 2018. A limitation of this dataset is that the data samples created by network flow analysis are saved in files, and processing these files is a time-consuming activity since each file contains a significant number of data instances.

Dataset Name: The ADFA Linux Dataset (ADFA-LD) [10, 11]

Created by: University of New South Wales

Year Created: 2014

Description: This dataset consists of 10 attack vectors along with the traces of the other data instances but has a limited range of attacks such as Zero-day attacks, Stealth attack, C100 Webshell attack. The dataset was based on system call traces and thus has the limitation of detecting for zero-day attacks and malware-based attacks. This dataset is not suitable for detecting attacks where an attacker manually tries to get initial connection, privilege escalation and lateral movement.

Dataset Name: ISCXIDS2012 [12, 13]

Created by: University of New Brunswick

Year Created: 2012

Description: This dataset consists of detection of attacks such as DoS, DDoS, Brute-flows force, Infiltration. This dataset was constructed with IP flows that consist of network scenarios with intrusive activities and labelled data instances. The data captures network traffic of protocols such as HTTP, SMTP, SSH, IMAP, POP3, and FTP to detect network-based attacks. This dataset has the limitation of detecting network level attacks and thus cannot detect attacks that are happening inside of a host Unix machine as it does not detect and capture any logs of the machine.

Early Detection of Host-Based Intrusions in Linux Environment [1]

This paper studies host-based intrusion detection systems (HIDSs) that have been utilizing the Australian Defence Force Academy Linux Dataset (ADFA-LD). HIDSs have also been subjected to a variety of machine learning techniques to improve detection performance for high accuracy and low false alarm rate. However, the practical implementation of HIDS for real-time intrusion detection receives less attention. To solve this restriction, we present a machine learning-based HIDS that can perform early intrusion detection using the same ADFA-LD dataset. Only a small number of system calls, invoked by programs early in their execution, are evaluated for intrusion detection in the proposed HIDS. The results of the experiments reveal that it is possible to achieve detection performance that is comparable to techniques that utilize all system calls invoked during the whole execution of programs.

Evaluating Host-Based Anomaly Detection Systems: A Preliminary Analysis of ADFA-LD [2]

This paper studies host-based intrusion detection systems (HIDSs), particularly anomaly-based HIDSs, which have gotten a lot of attention recently. However, due to the significant advancement of computer systems, the previous data sets utilised for HIDS assessment have lost much of their relevance over time. To address this need, this paper utilizes the ADFA Linux data set (ADFA-LD) was published. This paper conducts a preliminary investigation of ADFA-LD in this work to extract important information for designing new host-based anomaly detection systems. Some common parameters, such as length, common pattern, and frequency, are analysed specifically against ADFA-LD, in accordance with the community's general concerns. The authors created a KNN-based HIDS that to be tested using ADFA-LD. The experimental findings reveal that, while acceptable performance for a few types of attacks may be achieved, there is still a long way to go in completely understanding the complicated behaviour of a modern computer system and, ultimately, realising more intelligent HIDSs.

A Modern Implementation of System Call Sequence-Based Host-Based Intrusion Detection Systems [3]

This paper discussed the fact that researchers in the syscall HIDS arena have built several complicated and powerful syscall-based models to act as anomaly detectors, matching advances in the machine learning community. These models usually have a high degree of accuracy while focusing on reducing the number of false positives. However, this paper shows that with each iteration of the suggested model, the industry moves further away from the context in which these models are designed to function. Further, the implementation space for anomaly detection models is shrinking as kernels grow more elaborate and hardened. Furthermore, due to the fast progress of operating systems and the resulting complexity, datasets that are decades old are no longer relevant. This paper seeks in evaluating the latest Linux kernel 5.7.0-rc1, and to bridge the gap between theoretical models and their intended application settings in this research. The authors look at the viability of syscall-based HIDS in current operating systems as well as the limits that HIDS developers face. They address how recent kernel improvements have rendered the traditional syscall trace gather method of constructing syscall table wrappers obsolete and offer a new strategy for generating data and positioning our detection model. Finally, they present preliminary findings from their model, which basically demonstrate that, depending on their complexity, in-kernel machine learning models are possible.

Threat Detection and Response in Linux Endpoints [4]

This paper demonstrates that using open-source technologies like Osquery and Elastic for building a host-based intrusion detection. The authors also present an in-house designed Endpoint Detection and Response (EDR) for Linux systems. Further, the authors state that the advantage of developing in-house EDR tools compared to commercial EDR solutions is that it gives both the expertise and the technological capabilities to detect and analyse security events. Furthermore, they go through the tool's design and the benefits it provides. In the proposed system, all endpoint logs are collected on a single server, which we use to do correlation between events on multiple endpoints and automatically detect risks like as pivoting and lateral movements. Lastly, they discuss the various types of attacks that our tool can identify.

Deep-Hook: A Trusted Deep Learning-Based Framework for Unknown Malware Detection and Classification in Linux Cloud Environments [5]

This paper discusses that the fact that antivirus software and even more sophisticated malware detection technologies, have limits when it comes to identifying new, undiscovered, and elusive malware. The authors present Deep-Hook, a trusted framework for detecting unknown malware in Linux-based cloud settings, in this study. Deep-Hook uses a trustworthy method to hook the VM's volatile memory and obtain the memory dump to find malware footprints while the VM is running. The memory dumps are converted into visual pictures, which are then examined by a convolutional neural network (CNN) classifier. The papers show an accuracy of up to 99.9% in their experimental assessment results that show Deep-ability Hook's to identify and classify unknown malware (including evasive malware like rootkits) quickly, efficiently, and correctly.

Design and Implementation of an Intrusion Detection System Using Extended BPF in the Linux Kernel [14]

This paper examines the content of packet headers and payloads to identify network breaches to produce an intrusion detection system (IDS). Previously, an IDS, such as Snort, a popular open-source IDS, was used with traditionally built as a software running in user space on a hardware server. With the inclusion of Extended BPF (eBPF) in the Linux kernel, the authors suggest that it is now possible to effectively inspect and filter incoming packets directly in the kernel. The paper develops and constructs an IDS with two sections that function together. The Linux kernel is used for the first section. It employs eBPF to do quick pattern matching to pre-drop many packets that are unlikely to fit any rule. The user space is where the second phase takes place. It looks over the packets left by the first portion to see which rules match them. The authors experimented using a modified version of Snort's registered ruleset that reveals that their IDS system can beat Snort's maximum throughput by a factor of three under many of the scenarios evaluated.

Proposed Work

For building a host-based intrusion detection system (HIDS) for Linux using osquery, the first step is to configure the osqueryd to log from certain number of tables with a frequency. This will ensure that the osquery daemon runs the query at the interval set and log its results in /var/log/osquery/osqueryd.results.log file.

Once the logging has been started, attacking the machine with various techniques allows the osquery daemon to log the activities through the tables queries to the log file.

This log file is then passed onto a python script that will check for any addition of log lines into the file and perform a sequence of checks using regular expression to check if the log is malicious in nature. If it is then it logs into a log file corresponding to the nature of its activity.

7 modules for HIDS system using osquery:

A) How osquery is configured
B) What logs are collected
C) What vulnerabilities are exploited
D) How lateral movement and privilege escalation is performed
E) How to detect initial connections and sessions
F) How to detect lateral movement
G) How to detect privilege escalation

Module Description

A) How osquery is configured

The logs generated by osquery is saved in /var/log/osquery/ directory and the logger plugin is the default, filesystem.

To query using event-based tables, disable_events and disable_audit is set to false.

The number of worker threads is set to 10, as to allow the osquery to query simultaneously.

The host_identifier is set to hostname, so that the same config files can be set on all machines where logging has to be done and also so that when these logs are aggregated from various machine, we know which logs belong to which machine.

The enable_bpf_events is set to true so that bpf tables can be queried. However, these only show log those events that are added, and not those events that are logged.

The enable_syslog is set to true so that syslog events can be queried.

The enable_file_events is set to true so that any file modifications that happen in specific directories can be identified.

The audit_allow_process_events is set to true so that processes that run can be analysed and checked for any suspicious activity.

B) What logs are collected

There are a few logs that are collected:

(1) Logs from bpf_socket_events table to identify connections from and to the target vulnerable VM

(2) Logs from process_open_sockets table to identify all active connections to and from the vulnerable VM
(3) Logs from file_events table to identify any and all file actions that occurred in a set of files or in a set of file paths and their children directories
(4) Logs from process_events to identify and all process that run on the target vulnerable VM
(5) Logs from syslog_events to identify sessions that are created on the target vulnerable VM
(6) Logs from processes table to identify any running processes that have been created from commands run inside of a docker container

C) What vulnerabilities are exploited

The vulnerabilities exploited to get a reverse shell are:

(1) Default credentials used to login to apache tomcat server admin console
(2) Get bash, dash, rbash, or sh reverse shells or bind shells or meterpreter shell

D) How lateral movement and privilege escalation is performed

The privileges/vulnerabilities for lateral movement were simulated as follows:

- Non-root user part of docker group—This is part of intended functionality, but this issue is raised if an attacker is able to get access user part of docker group.

  Using the -v parameter to mount host directories onto the docker allows changing of contents of any file in the host system.

- Lateral Tool Transfer (T1570)—This involves the transfer of tools such as Linux privilege escalation scripts and malicious scripts.

Some of the different ways to escalating privileges from getting access to host file from inside of docker are:

(1) Abuse Elevation Control Mechanism: Setuid (T1548.001)

SUID bit can be set on a variety of tools available on the machine, which can then be used by www-data user to escalate their privileges. A list of all the tools that can be used for this purpose is given in https://gtfobins.github.io/#+suid.

(2) Abuse Elevation Control Mechanism: Sudo and Sudo Caching (T1548.003)

This way of privilege escalation is done by giving a particular user to run certain tools as root using sudo. This requires that these permissions be given in a special file known as /etc./sudoers. A list of all the tools that can be used for this purpose is given in https://gtfobins.github.io/#+sudo. Adding “www-data ALL = (ALL) NOPASSWD: ALL” in /etc./sudoers gives www-data user to run all tools as root using sudo without inserting their password.

(3) Scheduled Task/Jobs—Systemd Timers (T1053.006 + T1057)

This can only be done using CVE-2020–27,352 as the following commands need to be run as root: “systemctl enable systemdprivesc.timer” and “systemctl start systemdprivesc.timer”. Here I am creating a new systemd timer and enabling and starting it so that this timer runs the bash command provided to get a reverse shell.

(4) Scheduled Task/Jobs—Cron (T1053.003 + T1548.001)

Cron jobs are a type of scheduled tasks in Unix environment where it will run the command specified at the given condition. Here /bin/bash is copied to /tmp directory and SUID is added to it every second.

(5) Hijack Execution Flow: Dynamic Linker Hijacking (T1574.006)—LD_PRELOAD

Setting up of LD_PRELOAD and using it for privilege escalation is given in https://www.hackingarticles.in/linux-privilege-escalation-using-ld_preload/. It requires that /etc./sudoers be edited where a user is given sudo permissions and that LD_PRELOAD environment is added.

(6) Creating of a new local root account (T1136.001)—Can be done using all three methods

This can be done by appening a formatted line into /etc./passwd file. One way of adding a new low user with root privileges is using script given in https://flast101.github.io/docker-privesc/. Another way is by running following command adds a local user with no password, who can impersonate the root user.

(7) Account Manipulation—SSH Authorized Keys (T1098.004)

This requires that the SSH authorized keys be created and adding public key in.ssh folder of the user wanting to impersonate. Also, this requires a openssh-server be installed as the target vulnerable VM does not have it.

(8) LXD—Can be done only using CVE-2020–27,352

As www-data user is part of lxd group, although lxd tool is not installed on the target vulnerable VM. We can use install lxd, therefore, only CVE-2020–27,352 is only possible.

(9) Modify Authentication Process—Pluggable Authentication Modules—pam_unix.so (T1556.003 + T1570)

pam_unix.so can be made in a separate ubuntu vm using the following github repo: https://github.com/zephrax/linux-pam-backdoor. Downloading this.so file and configuring allows us to use to escalate our privileges.

(10) Event Triggered Execution—Unix Shell Configuration Modification (T1546.004)

When root user runs bash, then /root/.bashrc is loaded by default. In this file location, we can put a one-liner bash reverse shell to get a reverse shell when the root user runs bash.

According to https://attack.mitre.org/techniques/T1546/004/, some other files that can be used for similar effect are: /etc./profile, /etc./profile.d, ~/.bash_profile, ~/.bash_login, ~/.profile, ~/.bash_profile, and ~/.bash_logout.

(11) Setting and misusing capabilities

Setting up capabilities allows a non-root user to use those capabilities. Here we are setting up capabilities for python3.9 and misusing it to escalate our privileges.

E) How to detect initial connections and sessions

There are three sources of logs that are used to analyse for connections and sessions:

(1) bpf_socket_events table
Logs generated using this method where the family is 2 gives a list of network connections. Using logs generated from this, we can identify open ports, bind connections, reverse connections, downloading of files and other connections.
One major drawback of the logs generated by this method is that they can't log meterpreter shells and reverse shells generated from meterpreter shells. To overcome this issue, we use process_open_sockets table with processes table.
Open ports can be identified in these logs where local_address is "0.0.0.0" and the remote_address is not listed. This can help us to detect if there are any ports that have been opened by any webserver or ssh or for bind connections or for any other purpose. This is the first check that is done which acts asa a precursor to bind connections.
Bind connections can be identified using this method when the local_address is "0.0.0.0", but the remote_address should be listed. This is done after open ports are checked for as all those events where remote_address is not listed cannot be a connection.
Reverse connections can be identified using this method when the local_address is "0.0.0.0", the remote_address is not listed and the path listed is one of the following: '/usr/bin/nc', '/usr/bin/telnet', '/usr/bin/bash', '/usr/bin/dash', '/usr/bin/sh', '/usr/bin/rbash', and '/usr/bin/perl'. The multiple values to check for the path is given as a reverse connection can be initiated using anyone of these tools. One drawback of these logs regarding reverse shells is that it cannot identify reverse shells that have been initiated by meterpreter shells using the "shell" command.
Downloading of files can be detected using this method where the path is either '/usr/bin/wget' or '/usr/bin/curl' as these are two available ways using which files can be downloaded on the target vulnerable VM.
A list of common paths are removed from logging so that any other interesting logs can be logged for forensics. This list of common paths include the following: '/usr/bin/whoopsie', '/usr/lib/firefox/firefox', '/usr/lib/systemd/systemd-resolved', '/usr/sbin/NetworkManager', '/snap/snapd/14066/usr/lib/snapd/snapd', and '/lib/systemd/systemd-resolved' as these are run very frequently so that when analysing logs during forensics, the process is not hampered by unnecessary logs events.
(2) process_open_sockets table with processes table

Logs generated by joining these two tables using the process IDs where family is 2 gives a list of active network connections. In these types of logs, we were able to identify the status of connection and also identify connections that are removed.

One major drawback of these logs is that they only capture active connections. If a connection is not active when this query is run, then it will not be able to log it. To overcome this issue, we use bfp_socket_events table. Using logs generated from this, we can identify meterpreter shells, open ports, bind connections, reverse connections, and other connections.

Meterpreter shells use a name consisting of 5 random letters, which is a combination of capital letters and lowercase letters. This can be checked for using regular expression. There are exceptions to this regular expression with "https", "cupsd" and "snapd" as these are also five letter words that matches this regular expression. Adding an AND logic where the name is not https gives us meterpreter connections.

Open ports can be identified using these logs where local_adress are "0.0.0.0", remote_port is "0.0.0.0" and remote_port is 0. Here the local_port can be anything, which will show which ports on the local machine is open.

Bind connections can be identified using these logs where fd is 4 and the name is either one of the following: 'nc', 'telnet', 'bash', 'dash', 'sh', 'rbash', 'perl'. I have used a variety of names as anyone of these can be used to initiate a bind connection to an open port on the target machine. In these types of logs, we can identify the different states of connections as they progress, this helps in forensics of when the connection was initiated, when it was in progress and when it was closed. However, in these logs bind connections are not dependent on identifying open ports.

Reverse connections can be identified using these logs where fd is 3 and the name is either one of the following: 'nc', 'telnet', 'bash', 'dash', 'sh', 'rbash', 'perl'. I have used a variety of names as anyone of these can be used to initiate a reverse connection to an open port on the attacker's machine. In these types of logs, we can identify the different states of connections as they progress, this helps in forensics of when the connection was initiated, when it was in progress and when it was closed.

A concatenation of name, remote_address and remote_port is done to identify and remove connections that are frequently established, to ease the process of analysing these logs at the time of forensics. These types of logs are stored in other_connections.log for forensics purpose.

(3) syslog_events table

These types of logs are brought into osquery by configuring rsyslog to forward all syslogs using the format provided under Linux Syslog in osquery docs website. With types of logs, we can identify the sessions that are being created for a particular user. This is done using regular expression on the message part of the logs received. Filtering messages by checking for the presence of the keywords 'session' and 'user' using regular expression, gives us a list of sessions created.

F) How to detect lateral movement

Lateral Movement that is performed in the given vulnerable VM by moving laterally to the docker present in the vulnerable VM. This can be achieved by an attacker after moving on to the machine using Apache WebLogic by either getting a reverse shell or a bind shell or a meterpreter shell. Lateral Movement to docker container by its very nature is not something that is a suspicious behaviour.

Lateral Movement can be identified by process_events table as it makes a list of all the commands that are run. The easiest way of identifying lateral movement would be to filter out any process that have the path as docker. However, these logs will also include those commands that are run to see those images or containers available and more. So, to address this issue, a regular expression can be used against cmdline part of this log where it is starts with docker followed by either of the following words: attach, exec and run. These are the three keywords that can be used to laterally move into a docker container.

This above-mentioned method is used only for identifying that a user has moved laterally. This method is not useful in identifying in what commands does a user run after moving laterally into a docker container.

To identify that a user has moved laterally and to identify the commands that the user is running inside of the docker container, analysing the parent process ID of those process IDs that run the command supplied by the user shows us that a specific process is used to run the actual docker container. Identifying this command helps us to identify that a user is moving laterally and also the commands that the user is running inside of the docker container. The commands that are run inside a docker container can be logged by making a list of all the process IDs that start a docker container or that start a bash or sh or rbash or dash shells from within the docker container. This command is identified by number of regular expressions against the log from process_events table as follows:

The first regular expression is that the current working directory (cwd) should start with "/run/containerd/io.containerd.runtime.v2.task/moby/".

The second regular expression is that the command line argument of the log (cmdline) should contain "/usr/bin/containerd-shim-runc-v2 -namespace.* -id.* -address /run/containerd/containerd.sock|/run/containerd/io.containerd.runtime.v2.task)".

The third regular expression is that the command line argument of the log (cmdline) should contain "runc–root".

The condition for these regular expressions is that the first expression must be satisfied, but either of the second or third regular expression may be satisfied.

The drawback of this method is that we miss some commands run by the user inside of a newly created script that is run on the host machine using CVE-2020–27,352 (cgroups).

To overcome this drawback, we can use processes table to check for any cmdline that includes the following path: /var/lib/docker/overlay2/, as files created for abusing CVE-2020–27,352 (cgroups) are stored on this path on the host machine (ie. the above-mentioned path is a path on host machine). This captures scripts that are created and run by user on host machine and gives us a wider view of what is being run by a user to check for any suspicious activity.

G) How to detect privilege escalation

Privilege Escalation Scripts

Before moving on detecting users that trying to escalate their privileges, we can try to identify different enumerations done to identify ways of escalating privileges. To enumerate for ways to escalate ones privileges, an attacker can run privilege escalation scripts. Multiple ways are available to identify these scripts.

(1) Using Filename

Attackers may use scripts straight from the internet, without changing the filename of the script to enumerate. Using file_events table to detect any events on file on the following directories: "/root/.ssh/%%", "/home/%/.ssh/%%", "/home/%%", "/root/%%", "/tmp/%%", and "/dev/shm/%%" were used to identify the files on whom to check the filenames of. The following list includes a list of some well-known privilege escalation scripts: 'bashark.sh', 'LinEnum.sh', 'linpeash.sh', 'linux-exploit-suggester.sh', 'linux-exploit-suggester-2.pl', 'linuxprivchecker.py', 'lse.sh', 'noir-private-i.sh', 'private-i.sh', and 'unix-privesc-check'.

The drawback with this method is that if the filename is modified, then this method becomes unsuccessful in detecting these privilege escalation scripts.

(2) Using File Hash

Some attackers may use scripts straight from the internet, but with a slight modification of the filename. Using file_events table to detect any events on file on the following directories: "/root/.ssh/%%", "/home/%/.ssh/%%", "/home/%%", "/root/%%", "/tmp/%%", and "/dev/shm/%%" were used to identify the files on whom to check the file hashes of. Using the file hash, ensures that regardless of the filename, that contents of the file are the same.

The following list includes the file hashes of the above given filename (in no particular order): 'a7cf44139e5c86c06a99fdb01c21e37efb5c8744', '26bbf01183c7aacf331f9ecdf694d44122e1a089', '994b8ac7c04a2714792a2bb2f390c2257d1322ea', '51c3176be341179418a83ec7bc2a1b89e40ad45c', '75c306e90e3a31877a2adeb5d39072180aafb87d', '28571c8e1107650cd1d764419e9d160d464fac0b', '0c511357c395c7585573067316ffa7157fa1ec26', '34abaf9a7196751dcc54876f25c6656b3835d8f8', '2b0496aa002c9e8c6b9fb1b3a11ad5c455ec3588', and 'd3e44fc93a665a29e145fac99144be66cee8fbb5'.

The drawback with this method is that as these scripts are updated, if the attacker uses a different version of the same script mentioned above, the file hashes may not match as the contents are not exactly same. Also a slight modification in the contents of the file can result in changing of the hash of the file.

(3) Using String Signatures

This method is the best amongst all the methods available, as this method uses yara_strings to identify if a specific set of characters are in a file, from a file which is added or modified in each list of paths. This is done using yara_events table. Identifying a particular string from the privilege escalation script and adding it in yara that checks for either of the strings, will help us to find the scripts even if there are some changes made to the script or the version changes, provided that the string checked for is not altered or deleted.

To add this yara.conf file to osqueryd, we have to insert a comma (,) after the last pack on osquery.conf and add the following: "yara": "/opt/edr/packs/yara.conf". Also, in the options section of the yara.conf file, adding a comma (,) after the last option and adding the following option to enable yara strings: "enable_yara_string": "true".

Further, there is another way of enumerating privilege escalation methods. This can be done using logs from process_events table. A module from Metasploit can be used check for different ways to escalate privileges. It is known as “multi/recon/local_exploit_suggester”. This module runs a sequence of commands to check for a list of vulnerabilities when a meterpreter session is passed to it. The issue with detecting a sequence of commands is that if the sequence is jumbled or altered in any way then, the detection fails.

For this reason, I have checked if a particular command from process_event logs matches with the following commands: lsattr -l /etc./passwd', 'grep abrt-hook-ccpp /proc/sys/kernel/core_pattern', rep abrt-hook-ccpp /proc/sys/kernel/core_pattern', 'uname -v', 'cat /proc/sys/user/max_user_namespaces', 'cat /proc/sys/kernel/unprivileged_userns_clone', '/usr/bin/python3 /usr/bin/apport-cli –version', 'sh -c kill -64 $$ && id', 'docker ps', '/bin/bash /usr/bin/ldd –version', 'dbus-send –system –print-reply –dest = org.blueman.Mechanism –type = method_call /org.freedesktop.Dbus.Introspectable.Introspect', '/bin/sh -c ldd –version', '/bin/sh /usr/bin/which juju-run', '/bin/sh -c command -v lastore-daemon && echo true', 'cat /proc/cpuinfo', '/bin/sh -c command -v nmcli && echo true', ' grep ^ii', 'dpkg -l ntfs-3 g', 'dpkg-query –list – ntfs-3 g', 'grep ^ID = /etc./os-release', '/bin/uname -r', 'cat /proc/sys/kernel/yama/ptrace_scope', 'cat /proc/sys/kernel/yama/ptrace_scope', "/bin/sh -c test -x '/reptile/reptile_cmd' && echo true", '/bin/sh -c command -v bash && echo true', "/bin/sh -c test -x '/usr/local/Serv-U/Serv-U' && echo true", and 'sudo –version', 'id -un', 'lsattr'.

I have tried my best to keep only that commands that don’t start with “/bin/sh -c” as these can be changed to “/bin/bash -c” or “/bin/rbash -c” or “/bin/dash -c”. Also, a few commands like “uname -m” and “uname -r” are removed from the above list as these commands were

seen in the logs even when I did not run it. Hence, to decrease false positive, the commands were removed.
yara_events can also be run to identify for CVE-2020–27,352 (cgroups) by identifying for specific strings in the file created regardless of the name of the file create as we know of the location of where the file is stored on the host machine. The path location is "/var/lib/docker/overlay2/%%". The %% symbol means that the yara_events will look at files in this directory and their subdirectories.
Meterpreter Payload file
Using file_events table, which generates hash of a file created, we can check if the hash matches to the sha1 hash of a meterpreter payload, c324913b88f2ce7a043b8d1bd97e93be40860d58.
Another better way to check for starting or containing or ending hex strings using yara_events table.
Detecting Privilege Escalation
As seen from Sect. 1 that there are numerous ways and variations of these ways for an attacker to escalate their privileges. Hence, creating rules that perfectly fit each variation is not possible. Hence, the approach undertaken is to better identify different ways regardless of the variations that are used.
This is done after detecting commands that are run inside the docker container or initiated by a command in docker container.
(1) Detecting Abuse of Elevation Control Mechanism: Setuid
As setting SUID bit requires running "chmod + x …", we can detect usage of chmod by from within the docker container. This is done using check_docker function.
(2) Detecting Abuse of Elevation Control Mechanism: Sudo and Sudo Caching
As setting up this requires modification of /etc./sudoers file, we can identify if any command run within docker container uses this file. This is done using check_docker function.
(3) Detecting Scheduled Task/Jobs—Systemd Timers
As this requires that.timer and.service file be added into /usr/lib/systemd/system/ directory, we can look out for. timer and.service. This is done using check_docker function.
(4) Detecting Scheduled Task/Jobs—Cron jobs
As this requires that crontab command be run, we can filter for this keyword from the commands that are run inside the docker container or initiated by a command in docker container. This is done using check_docker function.
(5) Detecting Hijack Execution Flow: Dynamic Linker Hijacking—LD_PRELOAD
As this requires that /etc./sudoers be edited where a user is given sudo permissions and that LD_PRELOAD environment is added, we can check for any the mention of the file name and also for LD_PRELOAD. This is done using check_docker function.
Detection of Creating of a new local root account
As this requires that /etc./passwd file be appended with or manipulated, we can check for the mention of /etc./passwd file from the commands that are run inside docker containers. This is done using check_docker function.
(6) Detection of Account Manipulation—SSH Authorized Keys
As this requires that ssh authorized keys be added in.ssh folder of the user that they want to access using ssh-rsa, we can check for any command run inside docker container that have these two keywords in it. Also, as openssh-server can be checked for in the commands run inside of docker container. This is done using check_docker function.
(7) Detection of privilege escalation using lxd
As this requires that lxd to be installed and can only be done using CVE-2020–27,352. So, checking for lxd and lxc in commands that are run from a docker container. This is done using check_docker function.
(8) Detection of Modify Authentication Process—Pluggable Authentication Modules—pam_unix.so
As this uses a particular location to which the.so file is copied, we can check for the file location "/usr/lib/x86_64-linux-gnu/security/" in the commands that are run inside the docker container. This is done using check_docker function.
(9) Detection of Event Triggered Execution—Unix Shell Configuration Modification
As some of the files where this misuse can happen are: ~/.bashrc, /etc./profile, /etc./profile.d, ~/.bash_profile, ~/.bash_login, ~/.profile, ~/.bash_profile, and ~/.bash_logout, so we can check if these paths are listed in any command that is run inside of docker container. This is done using check_docker function.
(10) Detection of Setting Up of Capabilities

As setcap command is required to set capabilities, this command can used to identify for any setting up of capabilities from within a docker container. This is done using check_docker function.

Conclusion

Host-based intrusion detection system using osquery can be used to detect attacks that are network-based attacks used to get initial access and detect endpoint-based attacks, such as lateral movement to docker container and privilege escalation. This help us to detect attack that are not only network-based, but also host-based attacks. Heuristic method of detecting attacks from osquery provides visibility for a variety of attacks. This method detects open ports, bind shells, reverse shells, and meterpreter shells in the category of initial access. In addition, this method can detect lateral movement into a docker container. Further, it can detect and check for malicious commands run inside of the docker container, this allows us to detect any changes done to the host machine from inside of the docker container for privilege escalation.

Funding This study was not funded by anyone.

Declarations

Conflict of Interest The authors declare that they have no conflicts of interest.

Ethical Approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Zhang X, Niyaz Q, Jahan F, Sun W. Early detection of host-based intrusions in Linux environment. IEEE Int Conf Electro Info Technol (EIT). 2020;2020:475–9. https://doi.org/10.1109/EIT48999.2020.9208245.
2. M. Xie and J. Hu, "Evaluating host-based anomaly detection systems: A preliminary analysis of ADFA-LD," 2013 6th International Congress on Image and Signal Processing (CISP), 2013, pp. 1711–1716, doi: https://doi.org/10.1109/CISP.2013.6743952.
3. Byrnes J, Hoang T, Mehta NN, Cheng Y. A modern implementation of system call sequence based host-based intrusion detection systems. In: 2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), 2020, pp. 218–225, doi: https://doi.org/10.1109/TPS-ISA50397.2020.00037.
4. Agarwal S, Sable A, Sawant D, Kahalekar S, Hanawal MK. Threat detection and response in Linux endpoints. In: 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS), 2022, pp. 447–449, doi: https://doi.org/10.1109/COMSNETS53615.2022.9668567.
5. Landman T, Nissim N. Deep-Hook: A trusted deep learning-based framework for unknown malware detection and classification in Linux cloud environments. Neural Netw. 2021;144:648–85. https://doi.org/10.1016/j.neunet.2021.09.019.
6. https://www.uptycs.com/blog/osquery-what-it-is-how-it-works-and-how-to-use-it
7. https://www.unb.ca/cic/datasets/ids-2018.html
8. Iman S, Lashkari AH, Ali AG. Toward Generating a new intrusion detection dataset and intrusion traffic characterization. In: 4th International Conference on Information Systems Security and Privacy (ICISSP), Purtogal, January 2018
9. https://www.unb.ca/cic/datasets/ids-2017.html
10. https://research.unsw.edu.au/projects/adfa-ids-datasets
11. Creech G, Hu J. Engineering & Information Technology, UNSW Canberra, UNSW, "Developing a High-Accuracy Cross Platform Host-Based Intrusion Detection System Capable of Reliably Detecting Zero-Day Attacks", 2014 University of New South Wales, Engineering & Information Technology, http://handle.unsw.edu.au/1959.4/53218.
12. https://www.unb.ca/cic/datasets/ids.html
13. Ali S, Hadi S, Mahbod T, Ghorbani AA. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Comput Security. 2012;31(3):357–74. https://doi.org/10.1016/j.cose.2011.12.012.
14. Wang S-Y, Chang J-C. Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel. J Netw Comput Appl. 2022. https://doi.org/10.1016/j.jnca.2021.103283.
15. http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html